an instruction that calls the virtual machine runtime and that is removed, or replaced, by the runtime before returning from the call site.

[0009] A multitasking virtual machine (MVM) introduces further complication to these mechanisms. In order to save processing and memory, an MVM aims at sharing as much of the runtime representation of a class as possible between tasks. Targets for sharing include the platform-independent code, the meta-data describing the class, and the native code produced by the dynamic compiler. Code re-writing techniques can be harmful when code, whether it is platform-independent or produced by a runtime compiler, is shared between multiple tasks. For instance, rewriting native code to remove a class initialization barrier is incorrect when that code is shared between multiple tasks, since tasks that have not initialized the corresponding class may subsequently be given access to that class without it being initialized first.

[0010] Furthermore, part of the runtime representation of a class cannot be shared among tasks, in particular, the storage for the class's variables, the initialization state of the class for a given task, or the objects used in the first-class representation of the class (e.g., in the Java programming language, an instance of java.lang.Class).
It is the role of the multitasking virtual machine to efficiently implement the level of indirection needed to access the task-specific part of classes, most prominently, the storage for classes' variables, that correspond to the task on behalf of which code is being executed.

[0011] Hence, there is a need for a method and apparatus for an efficient class initialization barrier that enables sharing between multiple tasks/virtual machines of both interpreted platform-independent code and the equivalent native code produced at runtime. Because access to the global variables of classes can be frequent, and because such access must take place only if a class is initialized, class initialization barriers must also be made efficient for the case of class's variable access. Lastly, accesses to a class's variable in cases where it is known that the class is already initialized, or being initialized, should not pay any costs related to the class barrier mechanism.

SUMMARY

[0012] A multitasking virtual machine associates each shared representation of a class with a table of references to task class mirror objects, plus one task class mirror object per task using that class (i.e., per task having at least loaded that class). A task class mirror object holds information that cannot be shared between tasks, including a status of the class initialization state for the corresponding task.

[0013] In one embodiment of the present invention, each task class mirror table includes two task class mirror object pointers per task, stored at entries called the initialized and the resolved entries. The values of these two entries encode the status of the class with respect to class resolution and initialization by the corresponding task.

[0014] Two possible arrangements of initialized and resolved entries for a given task are favored. In the first arrangement, the initialized and the resolved entries are stored at consecutive position in the table, so that the position of one entry for a given task can simply be computed from the other by adding or subtracting 1. In the second arrangement, all the initialized entries are stored contiguously, followed by all the resolved entries, so that the position of one entry for a given task can simply be computed from the other by adding or subtracting the number of tasks supported by the table (or, by adding half of the total number of entries).

[0015] In one embodiment of the present invention, the task class mirror table of initialization-less classes (as defined in [0006]) can be made of a single task class mirror object pointer per-task, in order to minimize the space consumed by task class mirror tables. When a virtual machine uses this space-saving technique, referred hereafter as the initialization-less optimization, it must use the second arrangement of initialized and resolved entry described in [0013] for the task class mirror table of non-initialization-less classes.

[0016] In one embodiment of the present invention, the system indexes the task class mirror table using a task identifier. To reduce the number of CPU cycle spent to execute class initialization barrier, an encoding of the identifier of a task as an offset from the beginning of a class task mirror table to the initialized entry associated with that task is stored in the descriptor of every thread running on behalf of that task.

[0017] In one embodiment of the present invention, the system initializes all the entries of the task class mirror table of a class to a null pointer upon creation of the shared representation of the class.

[0018] In one embodiment of the present invention, upon loading of class C by a task T the system sets a resolved entry of the task class mirror table associated with C to the task class mirror object that holds the representation of C that is private to T. The index to the resolved entry is computed from T's identifier. If the class C is identified initialization-less during class loading and the multi-tasking virtual machine implementation uses the initialization-less optimization, the task class mirror is set to the initialized state, and its pointer is stored at the entry corresponding to T in the task class mirror table associated with C.

[0019] In one embodiment of the present invention, the system examines the initialized entry of a task class mirror table associated with a class C, at the